Delegates

«It is said that inheritance is useful only in case of having Donald Trump as a father»

The problem with wrongly designed or wrongly documented inheritance is that:

«The interaction of inherited classes with their parents can be surprising and unpredicatable if the ancestor wasn't designed to be inherited from»

Delegates – Standard delegation

```
private val textView : TextView by lazy {
    val view = findViewById(R.id.tvName) as TextView
    view
}
```

```
var name: ViewState by Delegates.observable(ViewState()) { property: KProperty<*>, old, new ->
    updateView(new)
}
```

```
var name: ViewState by Delegates.vetoable(ViewState()) { property: KProperty<*>, old, new ->
    new != null
}
```

```
var name: ViewState by notNull()
```

Delegates – Property delegation

The general structure of delegated properties looks like this:

```
[val | var] <Property Name> : <Property Typ> by <Expression>
```

The term **Expression** after **by** presents the **delegate**. Calls to get() on the property will be forwarded to the delegate. Therefore the delegate has to provide a **get()** method **for immutable properties** and a **set()** method for **mutable properties**

```
public interface ReadOnlyProperty<in R, out T> {
    public operator fun getValue(thisRef: R, property: KProperty<*>): T
}
```

```
public interface ReadWriteProperty<in R, T> {
    public operator fun getValue(thisRef: R, property: KProperty<*>): T
    public operator fun setValue(thisRef: R, property: KProperty<*>, value: T)
}
```

Delegates – Class delegation

You can delegate **interface** methods of a class to another class. It‘s like inheritance, but with 2 major differences.

- You can delegate multiple classes
- You only share the interface methods, no other methods and variables

Delegates – Class delegation

The Java way:

```
interface Base1{
    fun get1(): String
}
```

```
interface Base2{
    fun get2(): String
}
```

```
class DerivedByInheritance(): Base1,Base2{
    override fun get1(): String = "get1"
    override fun get2(): String = "get2"
}
```

The Kotlin way:

```
class Base1Impl: Base1{
    override fun get1(): String = "get1"
}
```

```
class Base2Impl: Base2{
    override fun get2(): String = "get2"
}
```

```
class DerivedByDelegates(private val base1: Base1,private val base2: Base2): Base1 by base1, Base2 by base2
```

Delegates – Map Delegate

```
class MyMessagingService : FirebaseMessagingService() {

    override fun onMessageReceived(message: RemoteMessage?) {
        super.onMessageReceived(message)
        val data = (message?.data ?: emptyMap()).withDefault { "" }
        val title = data["title"]
        val content = data["content"]
        print("$title $content")
    }
}
```

Delegates – Map Delegate

```
class NotificationParams(val map: Map<String, String>) {
    val title: String by map
    val content: String by map
}
```

```
override fun onMessageReceived(message: RemoteMessage?) {
    super.onMessageReceived(message)
    val data = (message?.data ?: emptyMap()).withDefault { "" }
    val params = NotificationParams(data)
    print("${params.title} ${params.content}")
}
```

End of section: Delegates

Any questions ?